Technical Cybersecurity

A Real Stack

function-args.c

Simple program

- Note the hexspeak again
- Really simple, using unsigned ints
- Shows calling convention boilerplate
- Shows stack manipulation and registry use

function-args.c

```
1
2 unsigned int call(unsigned int a) {
3   unsigned int j = 0xcafed00d ;
4   unsigned int k = a;
5   return 0xcafebabe;
6 }
7
8 int main(int argc, char* argv[]) {
9   unsigned int i = 0xdeadc0de;
10  unsigned int retval = call(i);
11  return retval;
12 }
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
```

```
cclamb@ubuntu:~/Work/abi-playground $ gdb fa
Reading symbols from fa...done.
(gdb) b main
Breakpoint 1 at 0x4004c1: file function-args.c, line 9
(gdb) b call
Breakpoint 2 at 0x40049e: file function-args.c, line 3
(gdb) r
Starting program: /home/cclamb/Work/abi-playground/fa

Breakpoint 1, main (argc=1, argv=0x7fffffffdec8) at fu
9         unsigned int i = 0xdeadc0de;
(gdb) disas
Dump of assembler code for function main:
   0x00000000004004b2 <+0>:     push   rbp
   0x00000000004004b3 <+1>:     mov    rbp,rsp
   0x00000000004004b6 <+4>:     sub    rsp,0x20
   0x00000000004004ba <+8>:     mov    DWORD PTR [rbp-
   0x00000000004004bd <+11>:    mov    QWORD PTR [rbp-
=> 0x00000000004004c1 <+15>:    mov    DWORD PTR [rbp-
   0x00000000004004c8 <+22>:    mov    eax,DWORD PTR [
   0x00000000004004cb <+25>:    mov    edi,eax
   0x00000000004004cd <+27>:    call   0x400497 <call>
   0x00000000004004d2 <+32>:    mov    DWORD PTR [rbp-
   0x00000000004004d5 <+35>:    mov    eax,DWORD PTR [
   0x00000000004004d8 <+38>:    leave
   0x00000000004004d9 <+39>:    ret
End of assembler dump.
(gdb) disas call
Dump of assembler code for function call:
   0x0000000000400497 <+0>:     push   rbp
   0x0000000000400498 <+1>:     mov    rbp,rsp
   0x000000000040049b <+4>:     mov    DWORD PTR [rbp-
   0x000000000040049e <+7>:     mov    DWORD PTR [rbp-
   0x00000000004004a5 <+14>:    mov    eax,DWORD PTR [
   0x00000000004004a8 <+17>:    mov    DWORD PTR [rbp-
   0x00000000004004ab <+20>:    mov    eax,0xcafebabe
   0x00000000004004b0 <+25>:    pop    rbp
   0x00000000004004b1 <+26>:    ret
End of assembler dump.
(gdb)
```

makefile

```
1  CC=gcc
2  OBJ=function2.o function-args.o print.o err.o
3  CC_FLAGS=-no-pie -fno-stack-protector -z execsta
4
5  %.o: %.c
6    $(CC) -c -o $@ $< $(CC_FLAGS)
7
8  main: $(OBJ)
9    $(CC) -o f2 function2.o $(CC_FLAGS)
10   $(CC) -o fa function-args.o $(CC_FLAGS)
11   $(CC) -o print print.o $(CC_FLAGS)
12   $(CC) -o err err.o $(CC_FLAGS)
13
14 clean:
15   rm *.o f2 print err
16   rm -rf a.out
17
```

function-args.c

```
1
2  unsigned int call(unsigned int a) {
3    unsigned int j = 0xcafed00d ;
4    unsigned int k = a;
5    return 0xcafebabe;
6  }
7
8  int main(int argc, char* argv[]) {
9    unsigned int i = 0xdeadc0de;
10   unsigned int retval = call(i);
11   return retval;
12 }
```

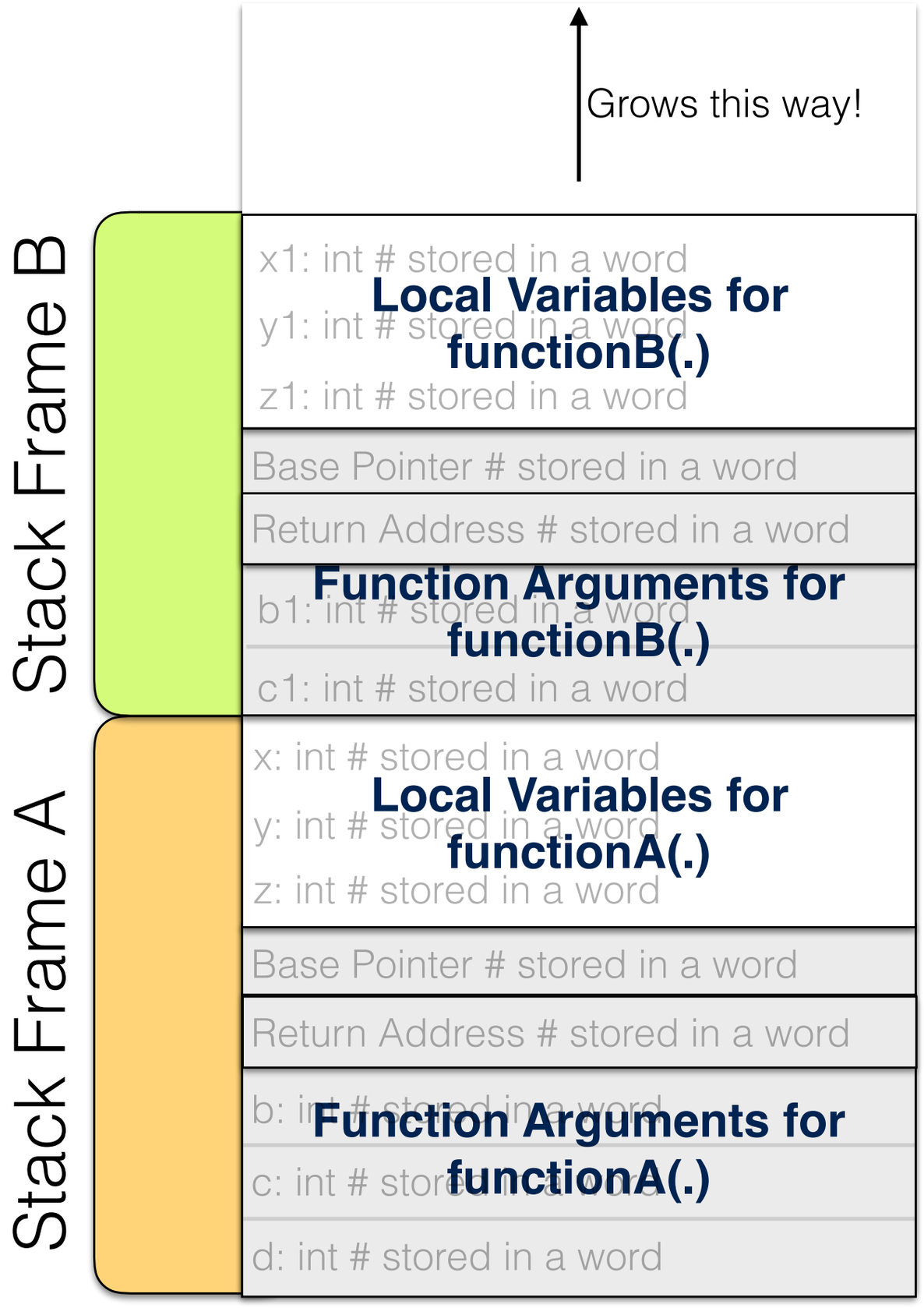
```
cclamb@ubuntu:~/Work/abi-playground $ gdb fa
Reading symbols from fa...done.
(gdb) b main
Breakpoint 1 at 0x4004c1: file function-args.c, line 9.
(gdb) b call
Breakpoint 2 at 0x40049e: file function-args.c, line 3.
(gdb) r
Starting program: /home/cclamb/Work/abi-playground/fa

Breakpoint 1, main (argc=1, argv=0x7fffffffdec8) at function-args.c:9
9         unsigned int i = 0xdeadc0de;
(gdb) disas
Dump of assembler code for function main:
   0x00000000004004b2 <+0>:     push   rbp
   0x00000000004004b3 <+1>:     mov    rbp,rsp
   0x00000000004004b6 <+4>:     sub    rsp,0x20
   0x00000000004004ba <+8>:     mov    DWORD PTR [rbp-0x14],edi
   0x00000000004004bd <+11>:    mov    QWORD PTR [rbp-0x20],rsi
=> 0x00000000004004c1 <+15>:    mov    DWORD PTR [rbp-0x4],0xdeadc0de
   0x00000000004004c8 <+22>:    mov    eax,DWORD PTR [rbp-0x4]
   0x00000000004004cb <+25>:    mov    edi,eax
   0x00000000004004cd <+27>:    call   0x400497 <call>
   0x00000000004004d2 <+32>:    mov    DWORD PTR [rbp-0x8],eax
   0x00000000004004d5 <+35>:    mov    eax,DWORD PTR [rbp-0x8]
   0x00000000004004d8 <+38>:    leave
   0x00000000004004d9 <+39>:    ret
End of assembler dump.
(gdb) disas call
Dump of assembler code for function call:
   0x0000000000400497 <+0>:     push   rbp
   0x0000000000400498 <+1>:     mov    rbp,rsp
   0x000000000040049b <+4>:     mov    DWORD PTR [rbp-0x14],edi
   0x000000000040049e <+7>:     mov    DWORD PTR [rbp-0x4],0xcafed00d
   0x00000000004004a5 <+14>:    mov    eax,DWORD PTR [rbp-0x14]
   0x00000000004004a8 <+17>:    mov    DWORD PTR [rbp-0x8],eax
   0x00000000004004ab <+20>:    mov    eax,0xcafebabe
   0x00000000004004b0 <+25>:    pop    rbp
   0x00000000004004b1 <+26>:    ret
End of assembler dump.
(gdb)
```

Data Flow

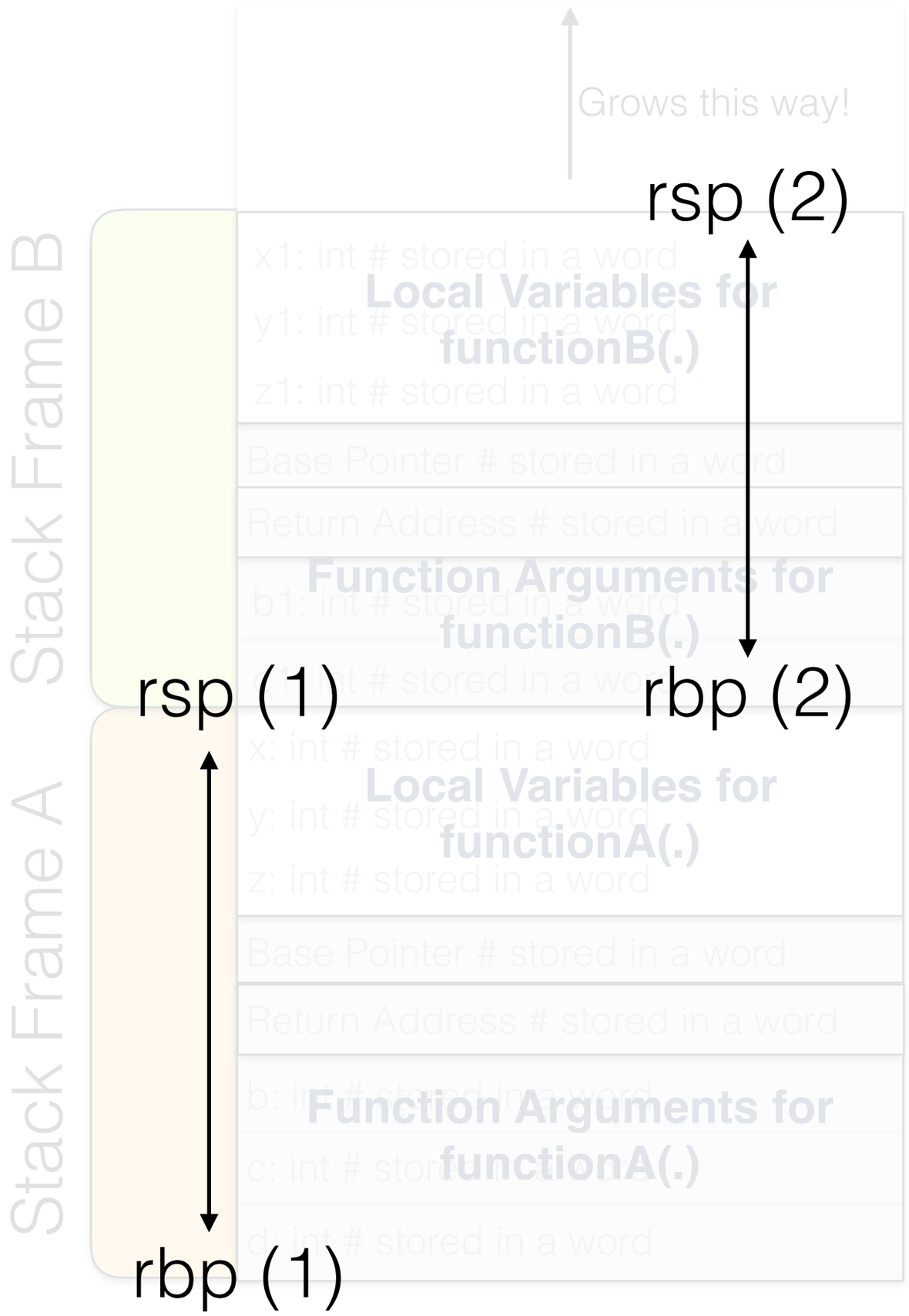
Registers

- rbp: bottom of stack
- rsp: top of stack
- edi: contains the argument



Stack Frames

```
push   rbp
mov    rbp,rsp
sub    rsp,0x20
mov    DWORD PTR [rbp-0x14],edi
mov    QWORD PTR [rbp-0x20],rsi
mov    DWORD PTR [rbp-0x4],0xdeadc0de
mov    eax,DWORD PTR [rbp-0x4]
mov    edi,eax
```



Accessing Data

```
tion call:
 push   rbp
 mov    rbp,rsp
 mov    DWORD PTR [rbp-0x14],edi
 mov    DWORD PTR [rbp-0x4],0xcafed00d
 mov    eax,DWORD PTR [rbp-0x14]
 mov    DWORD PTR [rbp-0x8],eax
 mov    eax,0xcafebabe
 pop    rbp
```

From call(.)

- reset the stack frame
- Pull data from the stack based on the bottom of the new stack
- Move arguments into memory from registers
- Move return value into register
- reset the base pointer

Next, let’s watch it run.